CHAPTER 3

▪▪▪

More Actions: A Galaxy of Possibilities

We hope you enjoyed making Evil Clutches and that it gave you a sense of how easy Game Maker is to use. However, you can achieve so much with a bit more knowledge, so let's move on to our second project and do something a little more adventurous.

Designing the Game: Galactic Mail

As before, it helps to set out a brief description of the game we want to create. We'll call this game *Galactic Mail* because it's about delivering mail in space. Here's the design:

You play an intergalactic mail carrier who must deliver mail to a number of inhabited moons. He must safely steer a course from moon to moon while avoiding dangerous asteroids. The mail carrier is paid for each delivery he makes, but pay is deducted for time spent hanging around on moons. This adds pressure to the difficult task of orienting his rickety, old rocket, which he cannot steer very well in space.

When the rocket is on a moon, the arrow keys will rotate it to allow the launch direction to be set. The spacebar will launch the rocket, and the moon will be removed from the screen to show that its mail has been delivered. In flight, the rocket will keep moving in the direction it is pointing in, with only a limited amount of control over its steering using the arrow keys. When things move outside the playing area, they reappear on the other side to give the impression of a continuous world. The player will gain points for delivering mail, but points will be deducted while waiting on a moon. This will encourage the player to move as quickly as possible from moon to moon. There will be different levels, with more asteroids to avoid. The game is over if the rocket is hit by an asteroid, and a high-score table will be displayed. Figure 3-1 shows an impression of what the final game will look like.

This description makes it possible to pick out all the various elements needed to create the game, namely moons, asteroids, and a rocket. For reasons that you will see later, we'll actually use two different moon objects (for a normal moon and an occupied moon) and two different rocket objects (for a "landed rocket" and a "flying rocket"). All the resources for this game can be found in the `Resources/Chapter03` folder on the CD.

Figure 3-1. *The Galactic Mail game features moons, asteroids, and a rocket ship.*

Sprites and Sounds

Let's begin by adding all the sprites to our game. In the previous chapter, we saw that sprites provide images for each element of the game. In this chapter, we'll use some extra abilities of sprites; however, before we can do this, you must set Game Maker into Advanced mode.

Setting Game Maker into Advanced mode:

1. If you are working on a game, you must save the game before switching modes.

2. Click the **File** menu and look for an item called **Advanced Mode**. If there is a check-mark in front of it, then you are already in Advanced mode. Otherwise, click that menu item to select it, and the main window should change to look like the one in Figure 3-2.

To make things simple, we'll leave Game Maker in Advanced mode for the remainder of the book, even though some of the options will only be used in the final chapters. Now we're going to start a new, empty game.

■**Note** To start a new game, choose **New** from the **File** menu. If you are already editing a game that has had changes made to it, you will be asked whether you want to save these changes.
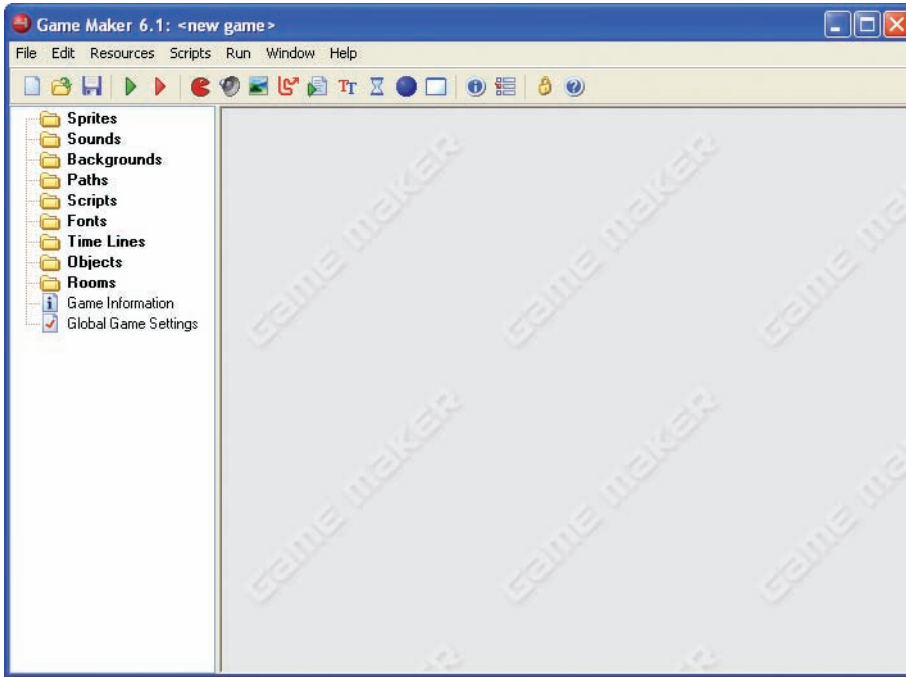

Figure 3-2. *In the main window of Game Maker in Advanced mode, there are a number of additional resources on the left and an additional menu.*

Our first step is to create all the sprites we need for the game. This works in the same way as in the previous chapter, but this time we must complete a couple of additional steps. Each sprite in Game Maker has its own *origin,* which helps to control the exact position in which it appears on the screen. By default, the origin of a sprite is set to be located at the top-left corner of the image. This means that when you move objects around in the game, it is as if you were holding them by their top-left corner. However, because the rockets in Galactic Mail need to sit in the center of the moons, it will be easier if we change the origin of all our sprites to be central.

Creating new sprite resources for the game:

1. From the **Resources** menu, choose **Create Sprite**. The Sprite Properties form with additional Advanced mode options will appear, like the one shown in Figure 3-3.

2. Click in the **Name** field and give the sprite a name. You should call this one `sprite_moon`.

3. Click the **Load Sprite** button. Select `Moon.gif` from the `Resources/Chapter03` folder on the CD.
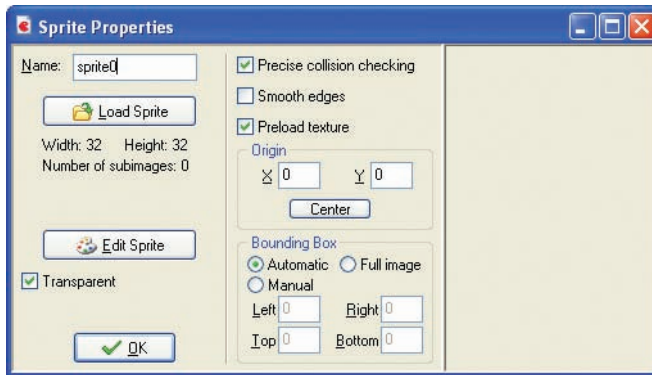
Figure 3-3. *This Sprite Properties form shows the advanced options.*

4. The controls for setting the origin are halfway down the second column of the form. Click the **Center** button to move the origin to the middle of the sprite. You should now see a cross in the middle of the sprite's image indicating the position of the origin. You can also change the origin by clicking on the sprite image with the mouse or typing in the **X** and **Y** values directly.

5. Enable the **Smooth edges** option by clicking on the box next to it. This will make the edges of the sprite look less jagged during the game by making them slightly transparent.

6. Click **OK** to close the form.

7. Now create asteroid and explosion sprites in the same way using `Asteroid.gif` and `Explosion.gif` (remember to center their origins too).

8. We'll need two sprites for the rocket: one for when it has landed on a moon and one for when it is flying through space. Create one sprite called `sprite_landed` using `Landed.gif` and another called `sprite_flying` using `Flying.gif`. Center the origins of these two sprites as before.

Before closing the Sprite Properties form for this last sprite, click the **Edit Sprite** button. A form will appear like the one shown in Figure 3-4. If you scroll down the images contained in this sprite, you'll see that it contains an animation of the rocket turning about a full circle. There are 72 different images at slightly different orientations, making up a complete turn of 360 degrees. We'll use these images to pick the correct appearance for the rocket as it rotates in the game. We can use the Sprite Editor to change the sprite in many ways, but for now simply close it by clicking the button with the green checkmark in the top left of the window.

Your game should now have five different sprites. Next let's add some sound effects and background music so that they are all ready to use later on.
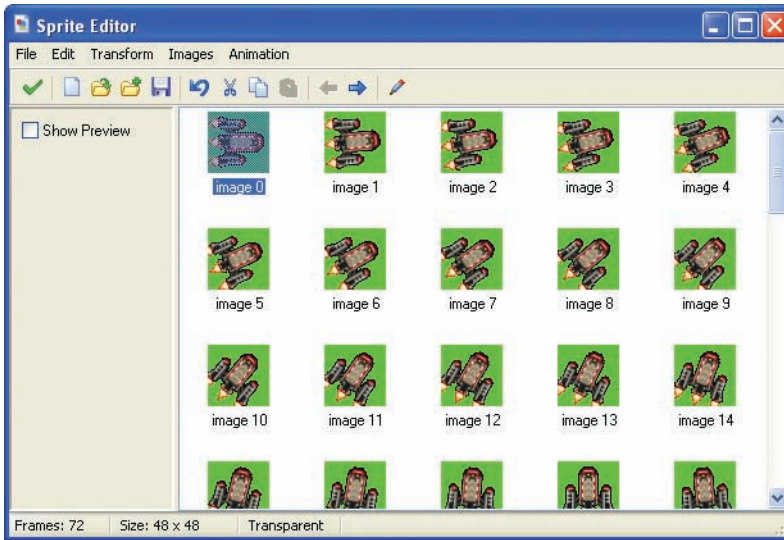
Figure 3-4. *The Sprite Editor shows all the images of the rocket.*

Creating new sound resources for the game:

1. Select **Create Sound** from the **Resources** menu. Note that the Sound Properties form now has additional Advanced mode options, but we don't need to worry about them for now (some of these are only available in the registered version of Game Maker).

2. Call the sound `sound_explosion` and click **Load Sound**. Select the `Explosion.wav` file from `Resources/Chapter03` on the CD.

3. Close the form by clicking **OK**.

4. Now create the `sound_bonus` and `music_background` sounds in the same way using the `Bonus.wav` and `Music.mp3` files.

Adding all these resources at the start will make it easier to drop them into the game as we are going along—so let's get started on some action.

Moons and Asteroids

Both moons and asteroids will fly around the screen in straight lines, jumping to the opposite side of the room when they go off the edge of the screen. In Game Maker this is called *wrapping*, and it is done using the **Wrap Screen** action.

Creating the moon object:

1. From the **Resources** menu, choose **Create Object**. The Advanced mode Object Properties form has additional options and actions too (see Figure 3-5).

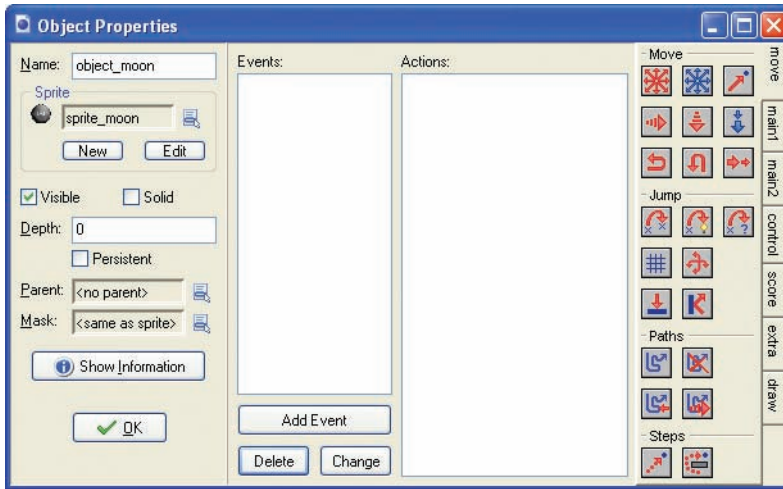2. Call the object `object_moon` and give it the moon sprite.

Figure 3-5. *The Object Properties form for the moon object looks like this.*

When a moon is created, we want it to start moving in a completely random direction.

Adding a create event to the moon object:

1. Click the **Add Event** button and choose the **Create** event.

2. Include the **Move Free** action in the **Actions** list for this event.

3. This action form requires a direction and a speed. Enter a **Speed** of 4 and type random(360) in the **Direction** property. This indicates a random direction between 0 and 360 degrees. The form should now look like Figure 3-6.
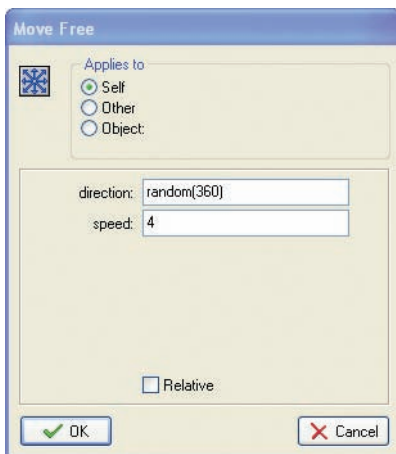

Figure 3-6. *Using the random command in a **Move Free** action will make the moons start moving in a random direction.*

We also need to make sure that when the moon goes off the edge of the room, it reappears at the other side.

Including a wrap action for the moon object:

1. Click the **Add Event** button, choose the **Other** events, and select **Outside Room** from the pop-up menu.

2. Include the **Wrap Screen** action in the **Actions** list.

3. In the form that appears, you should indicate that wrapping should occur in both directions (top to bottom and left to right). Now the form should look like Figure 3-7.

4. The moon object is now ready to go, so you can close the Object Properties form by clicking **OK**.

Figure 3-7. *The **Wrap Screen** action properties form looks like this.*

The asteroid object can be created in exactly the same way as the moon earlier. However, to keep things neat, we want to make sure that asteroids appear behind other objects when they cross paths with them on the screen. Instances of objects are usually drawn in the order in which they are created, so it is hard to be sure whether one type of object will appear in front of another. However, you can change this by setting an object's *depth* value. Instances with a smaller depth are drawn on top of instances with a larger depth, and so appear in front of them. All objects have a default depth of 0, so to make sure the asteroids appear behind other objects we simply give them a depth greater than 0.

Creating the asteroid object:

1. Create a new object called `object_asteroid` and give it the asteroid sprite.

2. On the left-hand side there is a text field labeled **Depth**. Enter `10` in this field to change the depth of the object from 0 to 10.

3. Add the **Create** event and include the **Move Free** action in the **Actions** list. Type `random(360)` in the **Direction** property and enter a **Speed** of `4`.

4. Add the **Other, Outside Room** event and include the **Wrap Screen** action in the **Actions** list (indicate wrapping in both directions).

■**Note** From now on we will use commas in event names, such as **Other, Outside Room** to show the two stages involved in selecting the event.

---

5. The Object Properties form should now look like Figure 3-8. Click **OK** to close the form.
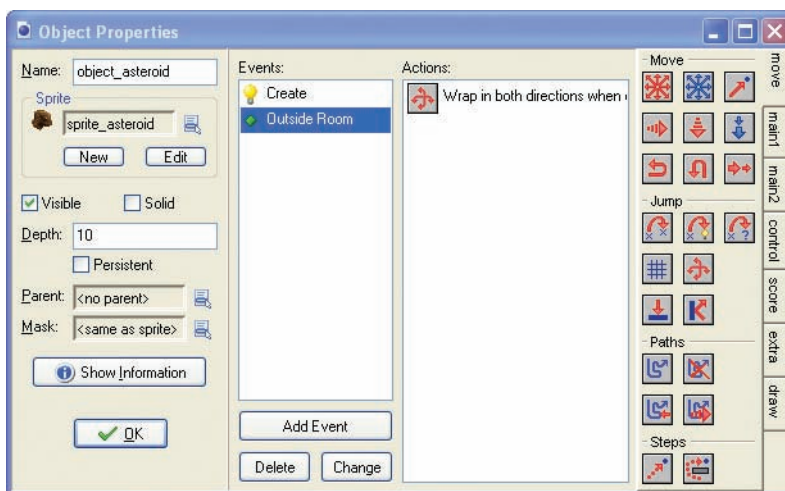

Figure 3-8. *We've set the depth for the asteroid object.*

Now would seem like a good time to check that everything has gone according to plan so far. However, before we can do that we must create a room with some instances of moons and asteroids in it.

Creating a room with moon and asteroid instances:

1. Select **Create Background** from the **Resources** menu.

2. Call the background `background_main`, and click the **Load Background** button. Select the `Background.bmp` image from the `Resources/Chapter03` folder on the CD.

3. Click **OK** to close the Background Properties form.

4. Select **Create Room** from the **Resources** menu. If the whole room isn't visible, then enlarge the window.

5. Select the **settings** tab and call the room `room_first`. Provide an appropriate caption for the room (for example "Galactic Mail").

6. Select the **backgrounds** tab. Click the menu icon to the right of where it says <no background> and select the background from the pop-up menu.

7. Select the **objects** tab and place a number of asteroids and moons in the room. (Remember that you can choose the object to place by clicking where it says "Object to add with left mouse"). The Room Properties form should now look like Figure 3-9.

8. Close the Room Properties form by clicking the green checkmark in the top-left corner.
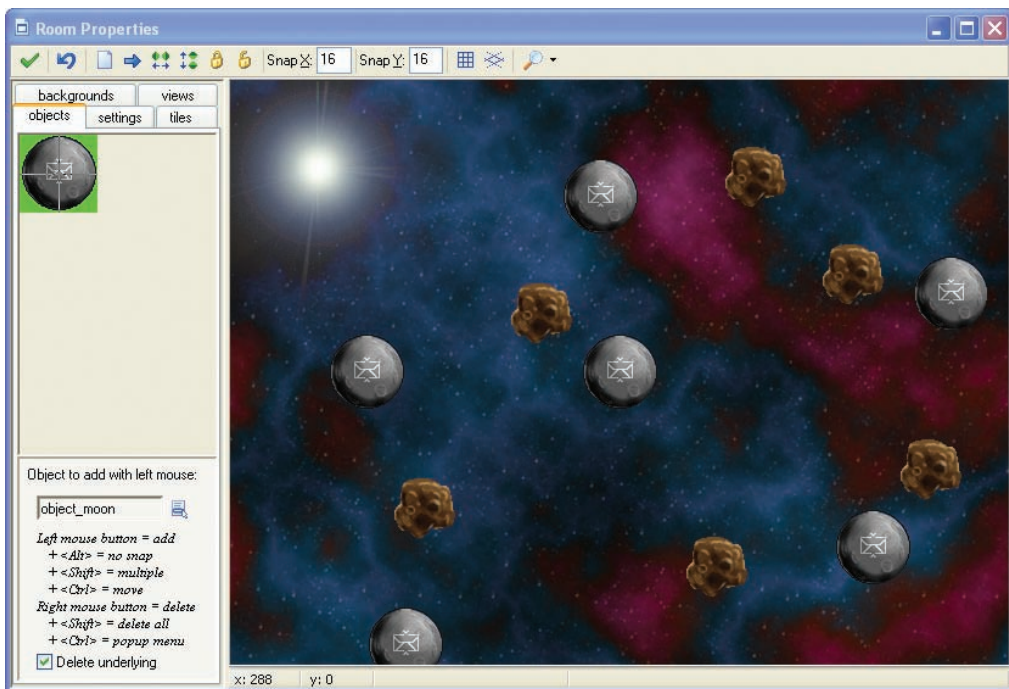

Figure 3-9. *Here's our first room.*

That should give us something to look at, so let's give it a try.

Saving and running the game:

1. Choose **Save** from the **File** menu (or click the disk icon). Save the game somewhere where you can easily find it again (the desktop, for example).

2. Select **Run normally** from the **Run** menu. If all goes well, the game should then appear in a new window.

Before continuing, double-check that everything is working the way it's supposed to. Are the moons and asteroids moving in different random directions? Do they reappear on the

other side of the screen when they leave the room? Do the asteroids always pass behind the moons? If any of these are not working, check that you have followed the instructions correctly. Alternatively, you can load the current version from the file `Games/Chapter03/galactic1.gm6` on the CD.

Flying Around

This isn't a very interactive experience yet, so let's introduce some gameplay by bringing the rocket into the game. We've already mentioned that we'll make two rocket objects, but let's stop to consider why this is necessary. Our rocket has two different ways of behaving: sitting on top of a moving moon with full control over the ship's direction, and flying through space with only limited control. Having two ways of controlling one object would involve a compli-cated set of events and actions, but if we separate these behaviors into two different objects, then it becomes quite simple. Provided that both objects look the same, the player will never notice that their ship is actually changing from being a "flying rocket" object to a "landed rocket" object at different stages of the game.

We also need two moon objects, as we want the landed rocket object to follow the path of one particular moon around (the one it has landed on). Making it into a separate object will allow us to single it out from the others in this way. As this second moon object will be almost the same as the normal moon, we can take a shortcut and make a copy of the existing moon object.

Creating the special moon object:

1. Right-click the moon object in the resource list, and select **Duplicate** from the pop-up menu. A copy of the moon object will be added to the resource list and its properties form is displayed.

2. Change the name to `object_specialmoon`. It is important that you use this exact name (including the underscore) as we will use this to identify this object later on.

3. Set the **Depth** of this object to `-5`. This will guarantee that instances of this moon are always in front of the other moons as it is lower than 0.

4. We will also make this moon responsible for starting the background music at the beginning of the game. Add an **Other**, **Game start** event and include a **Play Sound** action in it (**main1** tab). Select the background music sound and set **Loop** to true so that the music plays continuously.

5. Click **OK** to close the properties form.

Now open the first room and add a single instance of this new special moon to the level. Run the game and the music should play. (You won't notice any other difference because the special moon should look and behave exactly like the other moons.)

Now we can make our two rocket objects. We'll begin with the landed rocket, which needs to sit on the special moon object until the player decides to blast off. We'll use a **Jump Position** action to make it follow the special moon's position as it moves around the screen.

Creating the landed rocket object:

1. Create a new object called `object_landed` and give it the landed rocket sprite. Set the **Depth** to `-10` so that it appears in front of the moons and looks like it's sitting on the surface of the special moon.

2. Add a **Step, End Step** event to the new object. An **End Step** allows actions to be performed immediately before instances are drawn at their new position on the screen. Therefore, we can use this event to find out where the special moon has been moved to and place the rocket at the same location—just before both of them are drawn.

■**Note** A **Step** is a short period of time in which everything on the screen moves a very small distance. Game Maker normally takes 30 steps every second, but you can change this by altering the **Speed** in the **settings** tab for each room.

---

3. Include the **Jump Position** action in the **Actions** list for this event. This action allows us to move an object to the coordinates of any position on the screen. Type `object_specialmoon.x` into the **X** value and `object_specialmoon.y` into the **Y** value. These indicate the `x` and `y` positions of the special moon. Make sure that you type the names carefully, including underscores and dots (i.e., periods or full stops) in the correct positions. The action should now look like Figure 3-10.
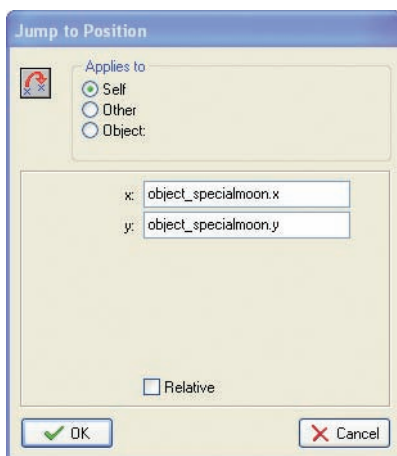

Figure 3-10. *We set the rocket to jump to the x and y positions of the special moon, so that it will follow this moon around.*

4. You might want to test the game now. Place one instance of the rocket at a random position in the room and run the game. The rocket should jump to the position of the special moon and stay on top of it as it moves around.

When you run the game, you will also notice that the rocket continually spins around without any user input. This is because the rocket sprite contains an animation showing the rocket rotating through 360 degrees. By default, Game Maker automatically cycles through a sprite's subimages to create an animation. However, this is not what is needed for this game—we need Game Maker to select the appropriate subimage based on the direction the rocket is moving in.

This requires a small amount of mathematics. There are 72 images representing a turn of 360 degrees, so each image must have been rotated by 5 degrees more than the last (because 360/72 = 5). Game Maker stores the direction of all objects in degrees, so it can work out which rocket subimage to use by dividing the rocket object's current direction by 5. Therefore we can make the rocket face in the right direction by using this rule (direction/5) to set the current subimage in a **Change Sprite** action.

Including a change sprite action in the landed object:

1. With the landed rocket Object Properties form open, include a **Change Sprite** action (**main1** tab) in its **End Step** event. Choose the landed rocket sprite from the menu and type `direction/5` into the **Subimage** property. `direction` is a special term that Game Maker recognizes as meaning the direction that this instance is currently facing in. Finally, set **Speed** to `0` to stop the sprite from animating on its own and changing the subimage. Figure 3-11 shows how this action should now look.

Figure 3-11. *Set the correct subimage in the sprite.*

■**Note** This way of dealing with rotated images might seem rather clumsy, but many old arcade games were made in a similar way so that each rotated image could include realistic lighting effects. Nonetheless, the registered version of Game Maker contains an additional action to rotate a sprite automatically without the need for subimages at all.

We can also make use of this special term for the object's direction to add actions that allow the player to control the direction of the rocket using the arrow keys.

Including keyboard events for the landed rocket object:

    1. Add a **Keyboard, <Left>** event to the landed rocket object.

    2. Include the **Move Free** action and type `direction+10` in the **Direction** property. This indicates that the current direction should be increased by 10 degrees. Set **Speed** to `0` because we don't want the rocket to move independently of the special moon. This action should now look like Figure 3-12.
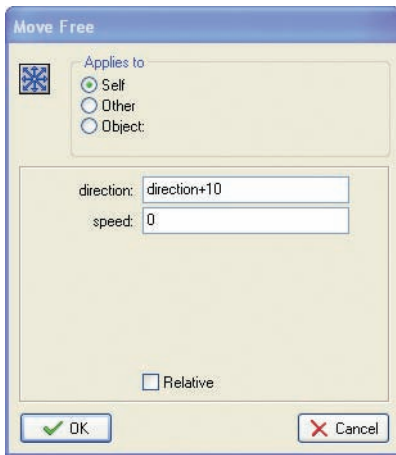

Figure 3-12. *Set the direction to equal itself plus 10.*

    3. Add a similar **Keyboard** event for the **<Right>** key. Include a **Move Free** action and type `direction-10` in the **Direction** property.

The last control we need for the landed rocket will allow the player to launch the rocket using the spacebar. This control will need to turn the landed rocket object into a flying rocket object, but we can't make an action for this as we haven't created the flying rocket object yet! So we'll make the flying rocket now and come back to this later.

Creating the flying rocket object:

    1. Create a new object called `object_flying` and select the flying rocket sprite. Set **Depth** to `-10` to make sure that this object appears in front of moons.

    2. Add an **Other, Outside Room** event and include a **Wrap Screen** action to wrap around the screen in both directions.

    3. Add an **End Step** event. Include a **Change Sprite** action, choose the flying rocket sprite, type `direction/5` in the **Subimage** property, and set the **Speed** to `0`.

4. Add a **Keyboard, <Left>** event and include a **Move Free** action. We don't want
the player to have too much control over the flying rocket, so type `direction+2` in
Direction and set **Speed** to 6.

5. Add a **Keyboard, <Right>** event with a **Move Free** action. Type `direction-2` in **Direc-
tion** and set **Speed** to 6.

The basic gameplay is nearly there now—just a few more events to tie up. First, the game
should end when the rocket hits an asteroid. Next, when the flying rocket reaches a moon, it
should turn into a landed rocket, and the moon should turn into the special moon (so that the
landed rocket can follow it). We achieve this using the **Change Instance** action, which basi-
cally turns an instance from one type of object into another. To return to our jelly comparison,
this is a bit like melting down the jelly from one instance and putting it into a new object
mold. Although the instance may end up as a completely different kind of object, it keeps
many of its original properties, such as its position on the screen and its direction. The fact
that these values remain the same is critical—otherwise the launch direction of the landed
rocket would get reset as soon as it turned into a flying rocket!

Adding collision events to the flying rocket object:

1. Add a **Collision** event with the asteroid object and include the **Restart Game** action
(**main2** tab) in the **Actions** list. Later on we'll include an explosion to make this more
interesting.

2. Add a **Collision** event with the moon object and include the **Change Instance** action
(**main1** tab). Set the object to change into `object_landed` using the menu button, and
leave the other options as they are.

3. Include a second **Change Instance** action for changing the moon into a special moon
object. To make this action change the moon object (rather than the rocket), we need
to switch the **Applies to** option from **Self** to **Other**. This makes the action apply to the
other object involved in the collision, which in this case is the moon. Set the object to
change into `object_specialmoon`. Figure 3-13 shows the settings.

Figure 3-13. *Change the other instance involved
in the collision into a special moon.*

Finally, we can go back to the landed rocket object. This will need an event that changes it into a flying rocket and deletes the special moon when the spacebar is pressed.

Adding a key press event to the landed rocket object:

1. Reopen the Object Properties form for the landed rocket by double-clicking on it in the resource list.

2. Add a **Key Press, <Space>** event and include a **Move Free** action to set the rocket in motion. Type `direction` in the **Direction** property (this keeps the direction the same) and set **Speed** to `6`.

3. Now include a **Change Instance** action and change the object into an `object_flying`.

4. Finally, we want to delete the special moon because it no longer needs to be visited. Include a **Destroy** action and change the **Applies to** option to **Object**. Click the menu button next to this and select the `object_specialmoon`, as shown in Figure 3-14.

Figure 3-14. *Include a **Destroy** action for the special moon.*

■**Caution** Using the **Object** setting for **Applies to** performs an action on all instances of that kind of object in the room. Deleting all of the special moon instances is fine in this case (as there is only one), but you will need to think carefully about the effects this setting will have before using it in your own games.

That completes the second version of our game. Make sure you save it and check that it all works as it should so far. You should now be able to rotate the rocket on a moon, launch it with the spacebar, and steer through the asteroids to land on another moon. Moons should disappear as you visit them, and the game should restart if you hit an asteroid. If something is not working, then check the instructions again, or compare your version with the version on the CD (`Games/Chapter03/galactic2.gm6`).

There are clearly a number of things still missing from the game, but the game is already quite fun to play. In the next section, we will add a scoring mechanism and a high-score table, as well as advancing the player to a new level once mail has been delivered to all the moons.

Winning and Losing

In this section we'll put a bit more effort into what happens when the player wins or loses the game. Let's begin by making asteroids a bit more explosive!

An Explosion

To get this working, we'll add a new explosion object and create an instance of it when the rocket hits an asteroid. This will play the explosion sound when it is created and end the game with a high-score table after the explosion animation has finished.

Adding an explosion object to the game:

1. Create a new object called `object_explosion`, and select the explosion sprite. Give it a **Depth** of `-10` to make it appear in front of other instances.

2. Add a **Create** event and include a **Play Sound** action (**main1** tab) for the explosion sound.

3. Add an **Other, Animation End** event. This event happens when a sprite reaches the final subimage in its animation.

4. Include the **Show Highscore** action (**score** tab) in the **Actions** list for this event. To make the high-score list look more interesting, set **Background** to the same as the background for the game, set **Other Color** to yellow, and choose a different font (e.g., Arial, bold). The action should now look like Figure 3-15.
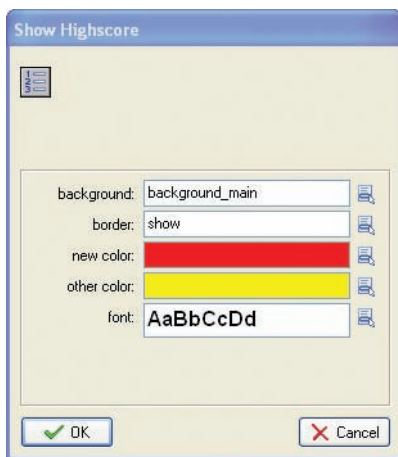

Figure 3-15. *You can spice up the high-score table.*

5. Also include a **Restart Game** action to start the game again after the high-score table is closed (**main2** tab).

6. Click **OK** to close the object.

Next we have to change the behavior of the flying rocket when it hits an asteroid.

Editing the flying rocket object:

1. Reopen the properties form for the flying rocket object by double-clicking on it in the resource list.

2. Select the **Collision** event with the asteroid by clicking on it once. Click once on the **Restart Game** action and press the Delete key to remove it from the action list.

3. Include a **Create Instance** action (**main1** tab) in its place, and set it to create the explosion object. Make sure the **Relative** property is enabled so that the explosion is created at the current position of the rocket.

4. Include a **Destroy Instance** action (**main1** tab) and leave it set to **Self** so that the rocket gets deleted. Click **OK** on the properties form to finish.

You might want to run the game now to see how it looks. Try colliding with an asteroid and you should get an explosion followed by the high-score table. Unfortunately, you can't score any points yet, so let's add this next.

Scores

If you've played the game quite a bit already, then you may have noticed a way of "cheating." You can avoid the risk of hitting asteroids by waiting for another moon to fly right next to your own and then quickly hop between moons. The game can become a lot less fun once this technique has been discovered, so our scoring system is designed to discourage the player from playing this way. Although they receive points for delivering mail, they also lose points for waiting on moons. This means that a player that takes risks by launching their rocket as soon as possible not only will have the most enjoyable playing experience but will also score the most points.

Editing game objects to include scoring:

1. Reopen the properties form for the special moon object and select the **Game Start** event. Include a **Set Score** action with a **New Score** of 1000. This gives the player some points to play with at the start. Close the properties form.

2. Reopen the properties form for the landed rocket and select the **End Step** event. Include a **Set Score** action with **New Score** as -1 and the **Relative** option enabled. This will repeatedly take 1 point off the score for as long as the player remains on a moon. As there are 30 steps every second, they will lose 30 points for every second of hanging around. Close the properties form.

3. Reopen the properties form for the flying rocket and select the **Collision** event with the moon object. Include a **Set Score** action with a **New Score** of 500 and the **Relative** option enabled.

4. Include a **Play Sound** action after setting the score and select the bonus sound.

Levels

At the moment, there is no reward for delivering all the mail. In fact, once all the moons are removed, the rocket just flies through space until it collides with an asteroid! This seems rather unfair, and it would be much better if the player advanced to a more difficult level. Making multiple levels in Game Maker is as simple as making new rooms. We can use actions to move between these rooms, and include more asteroids in the later levels to make them more difficult to play.

Let's begin by creating the new levels. You'll repeat these steps to make two more levels so that there are three in total. You can always add more of your own later on.

■**Note** The order of the rooms in the resource list determines the order of your levels in the game, with the top level being first and the bottom level last. If you need to change the order, just drag and drop them into new positions into the list.

Creating more level resources for the game:

1. Right-click on a room in the resource list and choose **Duplicate** from the pop-up menu. This will create a copy of the level.

2. Go to the **settings** tab and give the room an appropriate name (room_first, room_second, etc.).

3. Switch to the **objects** tab, and add or remove instances using the left and right mouse buttons.

4. Make sure that each level contains exactly one special moon and one instance of the landed rocket.

In order to tell Game Maker when to move on to the next room, we have to be able to work out when there are no moons left in the current one. To do this, we will use a *conditional action* that asks the question "Is the total number of remaining moons equal to zero?" If the answer is yes (or in computer terms, **true**), then a block of actions will be performed; otherwise the answer is no (or **false**), and this block of actions is skipped. We'll put this check in the collision event between the flying rocket and the moon, so that players complete the level as soon as they hit the final moon.

■**Note** All conditional actions ask questions like this, and their icons are octagon-shaped with a blue background so that you can easily recognize them.

Editing the flying rocket object to test for the number of remaining moons:

1. Reopen the properties form for the flying rocket and select the **Collision** event with the moon object.

2. At the end of the current list of actions, include the **Test Instance Count** action (**control** tab). Set the **Object** field to object_moon and the other settings will default to how we need them (**Number**, 0 and **Operation**, Equal to). This is now equivalent to the question "Is the total number of remaining moons equal to zero?" The form should look like Figure 3-16.
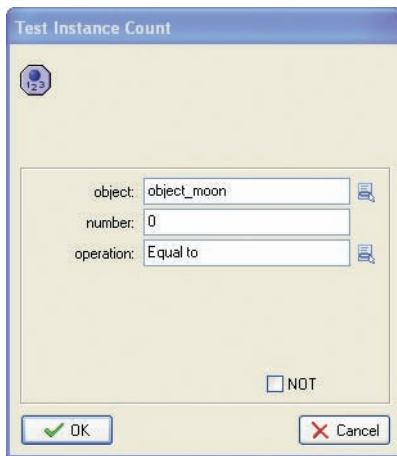

Figure 3-16. *We use the **Test Instance Count** action to count the number of moons.*

3. Below this action we need to start a *block*. A block indicates that a number of actions are grouped together as part of a conditional action. This means that all of the actions in the block will be performed if the condition is true and none of them if it is not. Add the **Start Block** action (**control** tab) directly below the condition to test the instances.

4. First, we will pause for a moment to give the player a chance to notice they have reached the final moon. Include the **Sleep** action (**main2** tab) and set **Milliseconds** to 1000. There are 1,000 milliseconds in a second, so this will sleep for 1 second.

5. We'll award the player an extra bonus score of 1,000 points when they finish a level. Include a **Set Score** action (**score** tab) with a **New Score** of 1000 and make sure that the **Relative** option is enabled.

6. Include the **Next Room** action from the **main1** tab to move to the next room. No properties need to be set here.

▽   **7.** Finally, add the **End Block** action (**control** tab) to end the block of the conditional action. The completed set of actions should now look like Figure 3-17. Note that the actions in the block are indented so that you can easily see that they belong together.
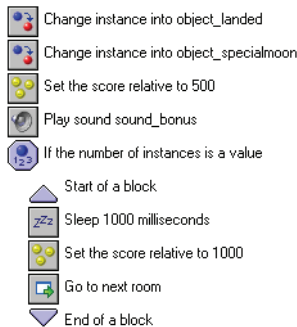

Figure 3-17. *Note that the actions in the block are indented.*

It is time to try out the game again. Save and play the game to check that you can go from one level to the next by visiting all the moons. You can also load this version of the game from the file `Games/Chapter03/galactic3.gm6` on the CD. However, if you complete the game you'll get an error message indicating that it has run out of levels. Don't worry—this is something we will fix in a moment, when we add some more finishing touches to the game.

Finishing Touches

To finish our game, we'll add an opening title screen, a help screen, and a congratulatory message upon completing the game. We'll also include a few visual touches to add a little bit of variety in the moons and asteroids.

A Title Screen

To create the title screen, we need a new object to display the name of the game and perform some initial tasks. We'll make it start the music and set the initial score, and then wait for the player to press a key before taking them to the first level.

Creating a new title object resource for the game:

1. Create a new sprite called `sprite_title` using `Title.gif`.

2. Create a new object called `object_title` and give it this sprite. Set the **Depth** property to `1` so that the moons go in front of it and the asteroids behind.

3. Add a **Create** event. This will contain the actions to start the music and set the score, but we've already created these in the special moon object, so we can simply move them over.

4. Open the special moon Object Properties form from the resource list and select the **Game Start** event to view its actions.

5. Drag and drop the two actions from the special moon **Game Start** event into the **Create** event of the title object. The **Game Start** event in the special moon should now be empty, and so it will delete itself automatically when the Object Properties form is closed. Do this now by clicking **OK** on the special moon's properties form.

6. Add a **Key Press, <Any key>** event to the title object and include the **Next Room** action in the action list for this event (**main1** tab).

Next we need to create a new room for the title screen.

Creating a new title room resource for the game:

1. Create a new room called `room_title` and give it an appropriate caption. Also set the room's background in the same way as before.

2. Add a few moon and asteroid instances to the room (just for effect).

3. Place an instance of the new title screen object in the center of the room.

4. Close the room properties.

5. To make sure that this is the first room in the game, drag the new room to the top of the list of rooms in the resource list.

Now quickly test the game to check that this all works correctly.

Winning the Game

We also need to stop the game from producing an error at the end and congratulate the player instead. Similar to how we created the title room, we will create a finish room with a finish object to display the message and restart the game.

Creating a new finish object resource for the game:

1. Create a new object called `object_finish`. It doesn't need a sprite.

2. Add a **Create** event to the object and include the **Display Message** action in it (**main2** tab). Set the **Message** to something like: "Congratulations! You've delivered all the mail."

3. Include a **Set Score** action, with a **New Score** of `2000` and the **Relative** option enabled.

4. Include the **Show Highscore** action, with **Background, Other Color**, and **Font** properties set as before.

5. Finally, include the **Restart Game** action.

Now that we have the object, we can create a room for it to go in.

Creating a new finish room resource for the game:

1. Create a new room and place one instance of the new finish object inside it. As this object has no sprite, it will appear as a blue ball with a red question mark on it. This will not appear in the game, but it reminds us that this (invisible) object is there when we are editing the room.

Now test the game to check that you can complete it—and that you get the appropriate message when you do (in other words, not an error message!)

Adding Some Visual Variety

At the moment all the moons look exactly the same, and the asteroids even rotate in unison as they move around the screen. However, with a different moon sprite and a little use of the random command, we can soon change this.

Editing the moon and asteroid objects:

1. Open the properties form for the moon object and click the **Edit** button below the name of the object's sprite (this is just another way of opening the moon sprite's properties).

2. In the moon sprite's properties, click **Load Sprite** and select `Bases.gif` instead of the existing sprite. This sprite contains eight subimages showing different kinds of inhabitations on each moon. Click **OK** to close the Sprite Properties form.

3. Back in the moon Object Properties form, select the **Create** event and include a new **Change Sprite** action. Select the moon sprite and type `random(8)` in the **Subimage** property. This will randomly choose one of the inhabited moon sprites. Also set **Speed** to `0` to stop the sprite from animating on its own and changing the subimage.

4. Close the Action Properties and the moon Object Properties forms.

5. Include an identical **Change Sprite** action to the **Create** event of the special moon object in the same way. There is no need to edit the moon sprite again, as both objects use the same one.

6. Open the properties form for the asteroid object and include a new **Change Sprite** action in its **Create** event as well. This time choose the asteroid sprite, and type `random(180)` in the **Subimage** property. There are 180 images in the rotating asteroid animation, so this will start each one at a different angle. Also type `random(4)` in the **Speed** property so that asteroids rotate at different speeds.

Help Information

Once you have finished making a game, it is easy to sit back and bask in your own creative genius, but there is one more important thing you must do before moving onto your next game. It may seem blindingly obvious to you how to play your masterpiece, but remember that it is rarely that obvious to a newcomer. If players get frustrated and stuck in the first few minutes because they can't figure out the controls, then they usually assume it is just a bad

game rather than giving it the chance it deserves. Therefore, you should always provide some help in your game to explain the controls and basic idea of the game. Fortunately, Game Maker makes this very easy through its **Game Information**.

Adding game information to the game:

1. Double-click on **Game Information** near the bottom of the resource list.

2. A text editor will open where you can type any text you like in different fonts and colors.

3. Typically you should enter the name of the game, the name of the author(s), a short description of the goals, and a list of the controls.

4. When you're done, click the green checkmark at the top left to close the editor.

That's all there is to it. When the player presses the F1 key during game play, the game is automatically paused until the help window is closed. Test the game one last time to check that this final version works correctly. You can also load the final version of the game from `Games/Chapter03/galactic4.gm6` on the CD.

Congratulations

Congratulations! You've now completed your second game with Game Maker. You might want to experiment with the game a bit further before continuing as there is much more you could do with it. To start with, you could make more levels with faster-moving asteroids or smaller moons to make it harder to land on them. We've included both larger planet sprites and smaller planetoids for you to experiment with, so see what you can come up with.

This chapter has introduced you to more features of Game Maker. In particular you've made use of events and actions to change sprites and objects. You've also used the **Depth** property of objects to control the order in which the instances appear on the screen. This chapter has also introduced *variables* for the first time, even though we haven't called them that yet. For example, the word `direction` is a variable indicating the current direction of an instance. We also used the variables `x` and `y` that indicate the position of an instance. There are many variables in Game Maker, and they are extremely useful. We will see plenty more of them in the chapters to follow.

In the next chapter, we'll continue to build on what you've learned so far by creating a crazy action game that requires quick thinking to avoid being squished. It's amazing what can go on in a deserted warehouse . . .